bdiGDB

JTAG debug interface for GNU Debugger

ARM7 / ARM9

User Manual

Manual Version 1.03 for BDI3000

abatron

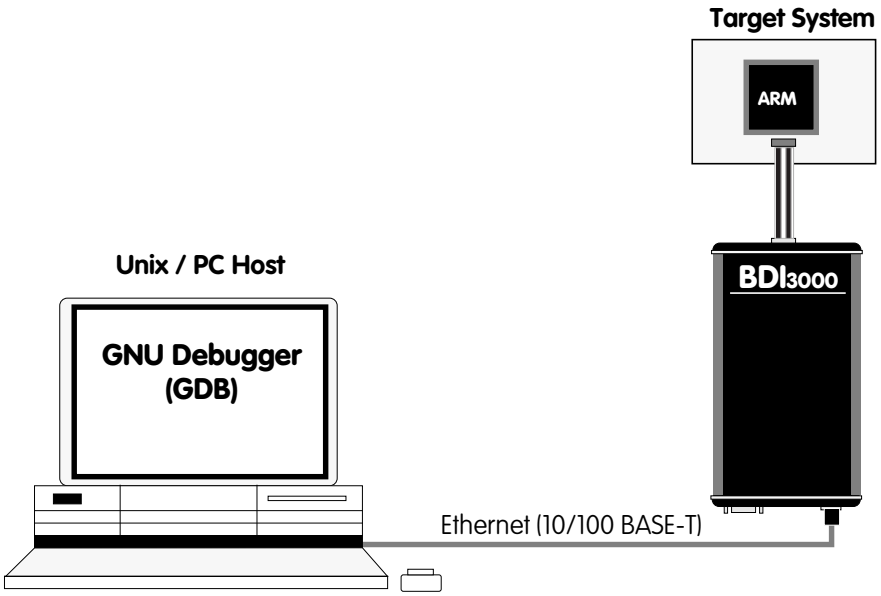
Appendices

1 Introduction

bdiGDB enhances the GNU debugger (GDB), with JTAG debugging for ARM7/ARM9 based targets. With the built-in Ethernet interface you get a very fast code download speed. No target communication channel (e.g. serial line) is wasted for debugging purposes. Even better, you can use fast Ethernet debugging with target systems without network capability. The host to BDI communication uses the standard GDB remote protocol.

An additional Telnet interface is available for special debug tasks (e.g. force a hardware reset, program flash memory).

The following figure shows how the BDI3000 interface is connected between the host and the target:



1.1 BDI3000

The BDI3000 is the main part of the bdiGDB system. This small box implements the interface between the JTAG pins of the target CPU and a 10/100Base-T Ethernet connector. The firmware of the BDI3000 can be updated by the user with a simple Linux/Windows configuration program or interactively via Telnet/TFTP. The BDI3000 supports 1.2 – 5.0 Volts target systems.

.

1.2 BDI Configuration

As an initial setup, the IP address of the BDI3000, the IP address of the host with the configuration file and the name of the configuration file is stored within the flash of the BDI3000.
Every time the BDI3000 is powered on, it reads the configuration file via TFTP.
Following an example of a typical configuration file:

```
; bdiGDB configuration file for ARM PID7T board
; ------------------------------------------
;
[INIT]
WM32   0x0B000020    0x00000000      ; Clear Reset Map
;
[TARGET]
CPUTYPE       ARM7TDMI
CLOCK         1             ;JTAG clock (0=Adaptive, 1=32MHz, 2=16MHz,...)
ENDIAN        LITTLE        ;memory model (LITTLE | BIG)
VECTOR        CATCH         ;catch unhandled exceptions
BDIMODE       AGENT         ;the BDI working mode (LOADONLY | AGENT)
BREAKMODE     SOFT          ;SOFT or HARD
;
[HOST]
IP            151.120.25.100
FILE          E:\cygnus\root\usr\demo\arm\myapp
FORMAT        COFF
LOAD          MANUAL        ;<AGENT> load application MANUAL or AUTO after reset

[FLASH]
WORKSPACE   0x00000000  ;workspace in target RAM for fast programming algorithm
CHIPTYPE    AM29F       ;Flash type (AM29F | AM29BX8 | AM29BX16 | I28BX8 | I28BX16)
CHIPSIZE    0x20000     ;The size of one flash chip in bytes (e.g. AM29F010 = 0x20000)
BUSWIDTH    8           ;The width of the flash memory bus in bits (8 | 16 | 32)
FILE        E:\cygnus\root\usr\demo\arm\boot.hex  ;The file to program
ERASE       0x04000000  ;erase sector 0 of flash in U12 (AM29F010)
ERASE       0x04004000  ;erase sector 1 of flash
ERASE       0x04008000  ;erase sector 2 of flash
ERASE       0x0400C000  ;erase sector 3 of flash
ERASE       0x04010000  ;erase sector 4 of flash
ERASE       0x04014000  ;erase sector 5 of flash
ERASE       0x04018000  ;erase sector 6 of flash
ERASE       0x0401C000  ;erase sector 7 of flash
```

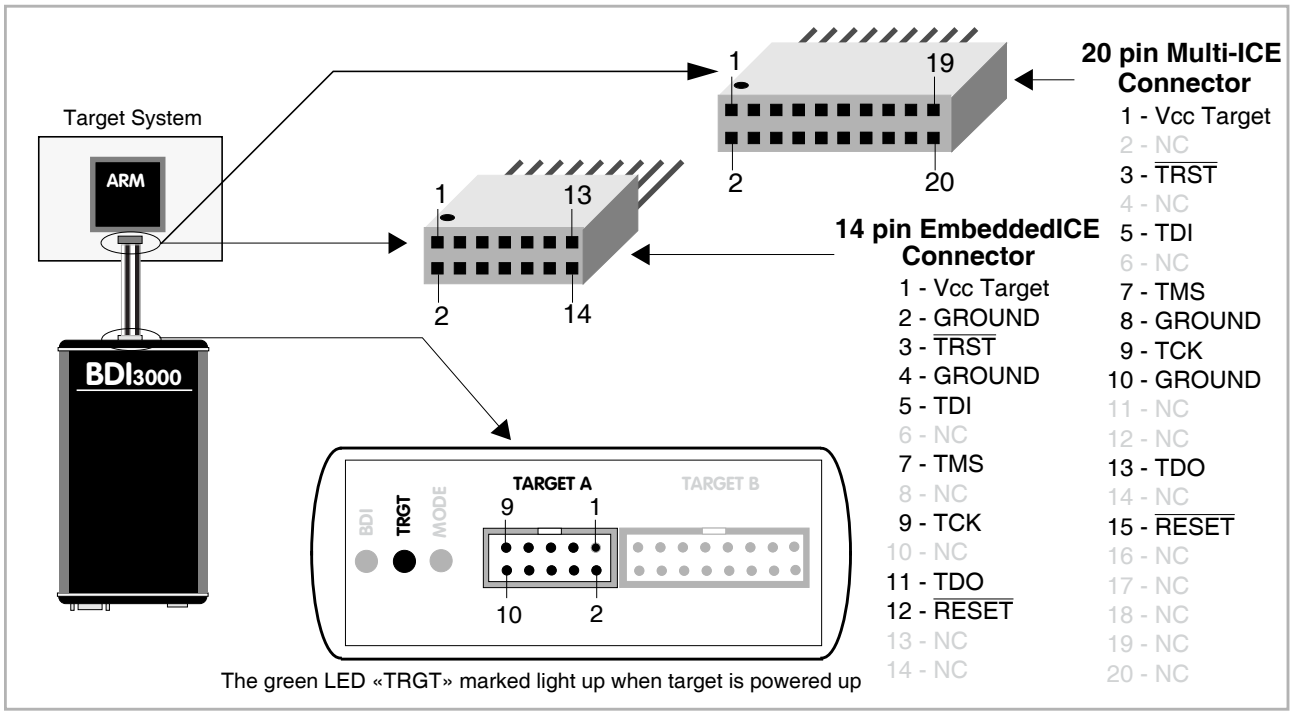
Based on the information in the configuration file, the target is automatically initialized after every reset.

2 Installation

2.1 Connecting the BDI3000 to Target

The enclosed cables to the target system are designed for the ARM Development Boards. In case where the target system has the same connector layout, the cable can be directly connected (14-pin EmbeddedICE or 20-pin Multi-ICE).

In order to ensure reliable operation of the BDI (EMC, runtimes, etc.) the target cable length must not exceed 20 cm (8").



For BDI MAIN / TARGET A connector signals see table on next page.

Warning:
Before you can use the BDI3000 with an other target processor type (e.g. PPC <--> ARM), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming a new firmware for an other target CPU.

BDI MAIN / TARGET A Connector Signals

Pin	Name	Description
1	reserved	This pin is currently not used.
2	$\overline{\text{TRST}}$	**JTAG Test Reset** This open-drain / push-pull output of the BDI3000 resets the JTAG TAP controller on the target. Default driver type is open-drain.
3+5	GND	**System Ground**
4	TCK	**JTAG Test Clock** This output of the BDI3000 connects to the target TCK line.
6	TMS	**JTAG Test Mode Select** This output of the BDI3000 connects to the target TMS line.
7	$\overline{\text{RESET}}$	This open collector output of the BDI3000 is used to reset the target system.
8	TDI	**JTAG Test Data In** This output of the BDI3000 connects to the target TDI line.
9	Vcc Target	**1.2 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board.
10	TDO	**JTAG Test Data Out** This input to the BDI3000 connects to the target TDO line.

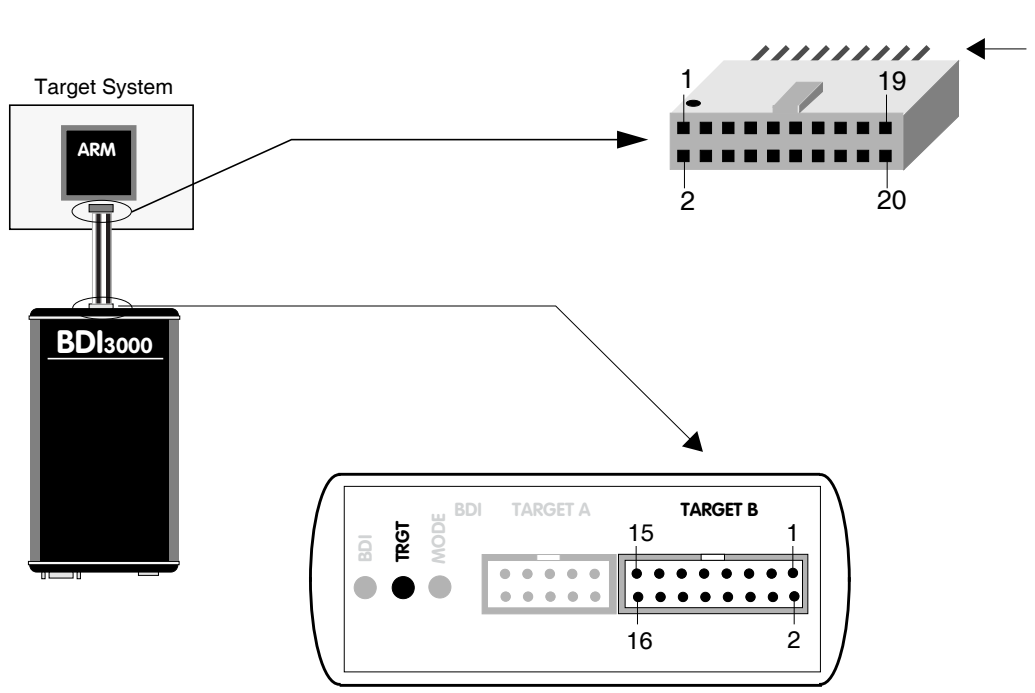
The BDI3000 works also with targets which have no dedicated $\overline{\text{TRST}}$ pin. For this kind of targets, the BDI cannot force the target to debug mode immediately after reset. The target always begins execution of application code until the BDI has finished programming the Debug Control Register.

Note:
For targets with a **14-Pin TI connector**, a special cable is available. This cable can be ordered separately from Abatron (p/n 90053).

2.1.1 Adaptive Clocking

Adaptive clocking is a feature which ensures that the BDI3000 never loses synchronization with the target device, whatever the target clock speed is. To achieve this, BDI3000 uses two signals TCK and RTCK. When adaptive clocking is selected, BDI3000 issues a TCK signal and waits for the Returned TCK (RTCK) to come back. BDI3000 does not progress to the next TCK until RTCK is received. For more information about adaptive clocking see ARM documentation.

Note:
Adaptive clocking is only supported with a special target cable (P/N 90052). This special cable can be ordered separately from Abatron.



The green LED «TRGT» marked light up when target is powered up

For TARGET B connector signals see table on next page.

Warning:
Before you can use the BDI3000 with an other target processor type (e.g. PPC <--> ARM), a new setup has to be done (see chapter 2.5). During this process the target cable must be disconnected from the target system.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming a new firmware for an other target CPU.

BDI TARGET B Connector Signals:

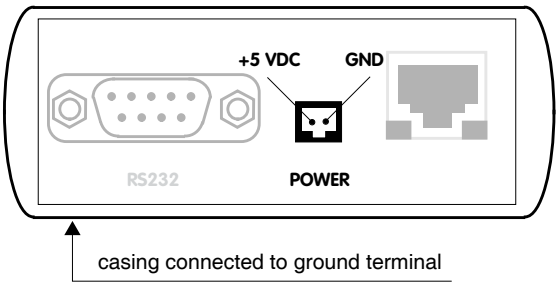
Pin	Name	Description
1	TDO	**JTAG Test Data Out** This input to the BDI3000 connects to the target TDO line.
2	reserved	
3	TDI	**JTAG Test Data In** This output of the BDI3000 connects to the target TDI line.
4	reserved	
5	RTCK	**Returned JTAG Test Clock** This input to the BDI3000 connects to the target RTCK line.
6	Vcc Target	**1.2 – 5.0V:** This is the target reference voltage. It indicates that the target has power and it is also used to create the logic-level reference for the input comparators. It also controls the output logic levels to the target. It is normally fed from Vdd I/O on the target board.
7	TCK	**JTAG Test Clock** This output of the BDI3000 connects to the target TCK line.
8	$\overline{\text{TRST}}$	**JTAG Test Reset** This open-drain / push-pull output of the BDI3000 resets the JTAG TAP controller on the target. Default driver type is open-drain.
9	TMS	**JTAG Test Mode Select** This output of the BDI3000 connects to the target TMS line.
10	reserved	
11	reserved	
12	GROUND	**System Ground**
13	$\overline{\text{RESET}}$	**System Reset** This open-drain output of the BDI3000 is used to reset the target system.
14	reseved	
15	reseved	
16	GROUND	**System Ground**

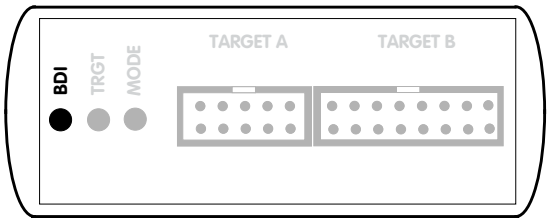
2.2 Connecting the BDI3000 to Power Supply

The BDI3000 needs to be supplied with the enclosed power supply from Abatron (5VDC).

Before use, check if the mains voltage is in accordance with the input voltage printed on power supply. Make sure that, while operating, the power supply is not covered up and not situated near a heater or in direct sun light. Dry location use only.

For error-free operation, the power supply to the BDI3000 must be between 4.75V and 5.25V DC. **The maximal tolerable supply voltage is 5.25 VDC. Any higher voltage or a wrong polarity might destroy the electronics.**





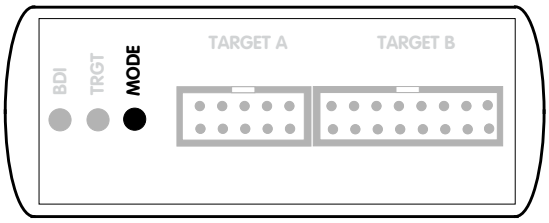
The green LED «BDI» marked light up when 5V power is connected to the BDI3000

Please switch on the system in the following sequence:

- 1 –> external power supply
- 2 –> target system

2.3 Status LED «MODE»

The built in LED indicates the following BDI states:



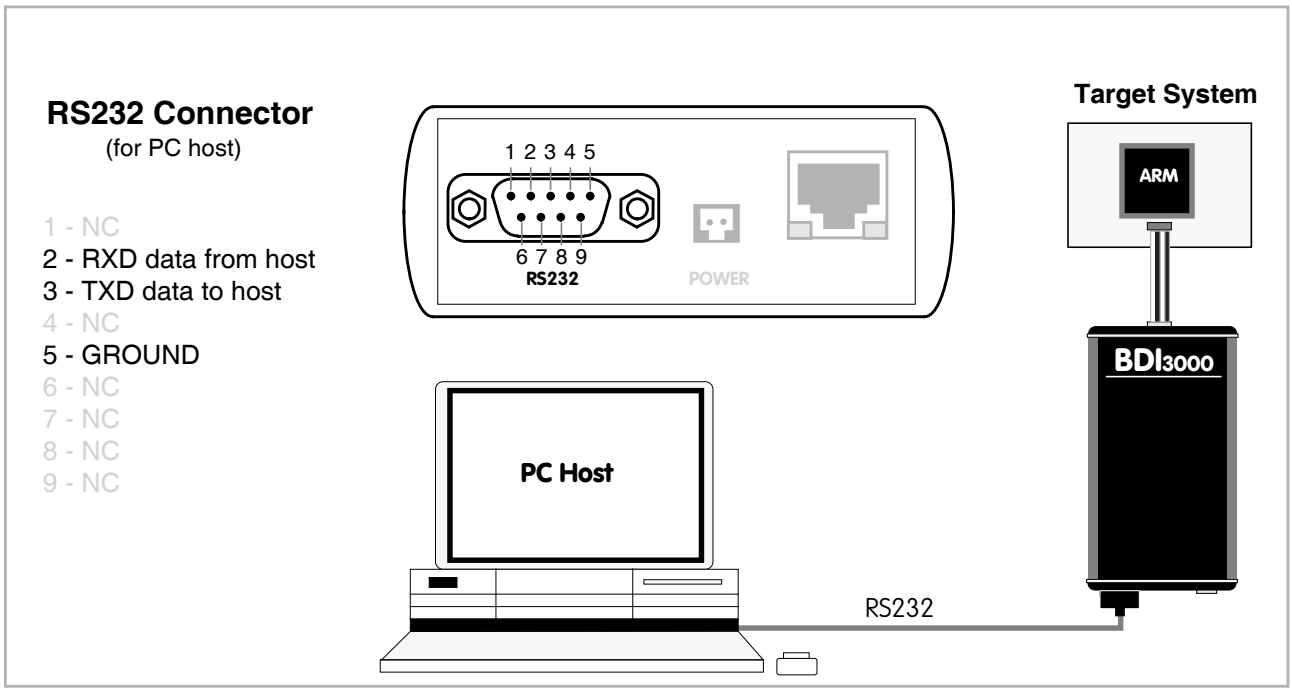
MODE LED	BDI STATES
OFF	The BDI is ready for use, the firmware is already loaded.
ON	The output voltage from the power supply is too low.
BLINK	The BDI «loader mode» is active (an invalid firmware is loaded or loading firmware is active).

2.4 Connecting the BDI3000 to Host

2.4.1 Serial line communication

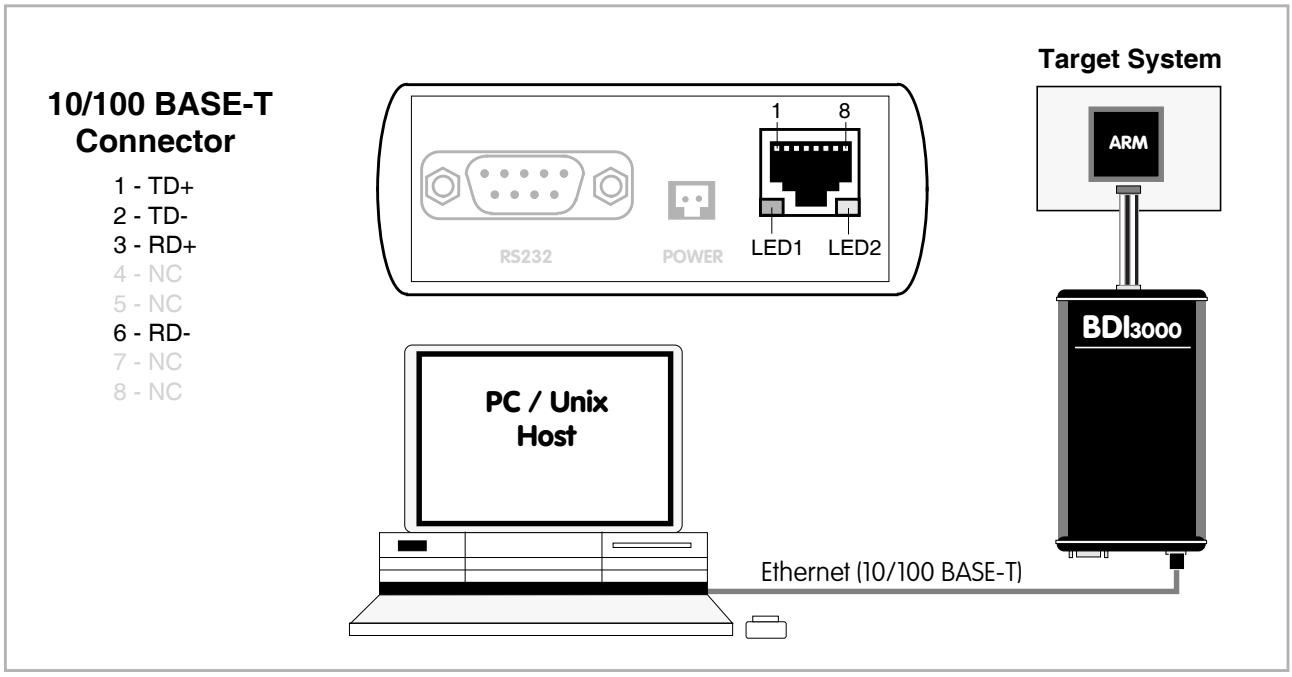
Serial line communication is only used for the initial configuration of the bdiGDB system.

The host is connected to the BDI through the serial interface (COM1...COM4). The communication cable (included) between BDI and Host is a serial cable. There is the same connector pinout for the BDI and for the Host side (Refer to Figure below).



2.4.2 Ethernet communication

The BDI3000 has a built-in 10/100 BASE-T Ethernet interface (see figure below). Connect an UTP (Unshielded Twisted Pair) cable to the BD3000. Contact your network administrator if you have questions about the network.



The following explains the meanings of the built-in LED lights:

LED	Function	Description
LED 1 (green)	Link / Activity	When this LED light is ON, data link is successful between the UTP port of the BDI3000 and the hub to which it is connected. The LED blinks when the BDI3000 is receiving or transmitting data.
LED 2 (amber)	Speed	When this LED light is ON, 100Mb/s mode is selected (default). When this LED light is OFF, 10Mb/s mode is selected

2.5 Installation of the Configuration Software

On the enclosed diskette you will find the BDI configuration software and the firmware required for the BDI3000. For Windows users there is also a TFTP server included.

The following files are on the diskette.

b30armgd.exe	Windows Configuration program
b30armgd.xxx	Firmware for the BDI3000
tftpsrv.exe	TFTP server for Windows (WIN32 console application)
*.cfg	Configuration files
*.def	Register definition files
bdisetup.zip	ZIP Archive with the Setup Tool sources for Linux / UNIX hosts.

Overview of an installation / configuration process:

- Create a new directory on your hard disk
- Copy the entire contents of the enclosed diskette into this directory
- Linux only: extract the setup tool sources and build the setup tool
- Use the setup tool or Telnet (default IP) to load/update the BDI firmware
  Note: A new BDI has no firmware loaded.
- Use the setup tool or Telnet (default IP) to load the initial configuration parameters
  - IP address of the BDI.
  - IP address of the host with the configuration file.
  - Name of the configuration file. This file is accessed via TFTP.
  - Optional network parameters (subnet mask, default gateway).

Activating BOOTP:
The BDI can get the network configuration and the name of the configuration file also via BOOTP. For this simple enter 0.0.0.0 as the BDI's IP address (see following chapters). If present, the subnet mask and the default gateway (router) is taken from the BOOTP vendor-specific field as defined in RFC 1533.

With the Linux setup tool, simply use the default parameters for the -c option:

```
[root@LINUX_1 bdisetup]# ./bdisetup -c -p/dev/ttyS0 -b57
```

The MAC address is derived from the serial number as follows:
MAC: 00-0C-01-xx-xx-xx , replace the xx-xx-xx with the 6 left digits of the serial number
Example: SN# 33123407 ==>> 00-0C-01-33-12-34

Default IP: 192.168.53.72
Before the BDI is configured the first time, it has a default IP of 192.168.53.72 that allows an initial configuration via Ethernet (Telnet or Setup Tools). If your host is not able to connect to this default IP, then the initial configuration has to be done via the serial connection.

2.5.1 Configuration with a Linux / Unix host

The firmware update and the initial configuration of the BDI3000 is done with a command line utility. In the ZIP Archive bdisetup.zip are all sources to build this utility. More information about this utility can be found at the top in the bdisetup.c source file. There is also a make file included.
Starting the tool without any parameter displays information about the syntax and parameters.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.

Following the steps to bring-up a new BDI3000:

1. Build the setup tool:
The setup tool is delivered only as source files. This allows to build the tool on any Linux / Unix host. To build the tool, simply start the make utility.

```
[root@LINUX_1 bdisetup]# make
cc -O2   -c -o bdisetup.o bdisetup.c
cc -O2   -c -o bdicnf.o bdicnf.c
cc -O2   -c -o bdidll.o bdidll.c
cc -s bdisetup.o bdicnf.o bdidll.o -o bdisetup
```

2. Check the serial connection to the BDI:
With "bdisetup -v" you may check the serial connection to the BDI. The BDI will respond with information about the current loaded firmware and network configuration.
Note: Login as root, otherwise you probably have no access to the serial port.

```
$ ./bdisetup -v -p/dev/ttyS0 -b115
BDI Type : BDI3000 (SN: 30000154)
Loader   : V1.00
Firmware : unknown
MAC      : ff-ff-ff-ff-ff-ff
IP Addr  : 255.255.255.255
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 255.255.255.255
Config   : ÿÿÿÿÿÿÿ........
```

3. Load/Update the BDI firmware:
With "bdisetup -u" the firmware is programmed into the BDI3000 flash memory. This configures the BDI for the target you are using. Based on the parameters -a and -t, the tool selects the correct firmware file. If the firmware file is in the same directory as the setup tool, there is no need to enter a -d parameter.

```
$ ./bdisetup -u -p/dev/ttyS0 -b115 -aGDB -tARM
Connecting to BDI loader
Programming firmware with ./b30armgd.100
Erasing firmware flash ....
Erasing firmware flash passed
Programming firmware flash ....
Programming firmware flash passed
```


4. Transmit the initial configuration parameters:
With "bdisetup -c" the configuration parameters are written to the flash memory within the BDI.
The following parameters are used to configure the BDI:

BDI IP Address	The IP address for the BDI3000. Ask your network administrator for assigning an IP address to this BDI3000. Every BDI3000 in your network needs a different IP address.
Subnet Mask	The subnet mask of the network where the BDI is connected to. A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask. If the BDI and the host are in the same subnet, it is not necessary to enter a subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value.
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI3000 after every start-up.
Configuration file	Enter the full path and name of the configuration file. This file is read via TFTP. Keep in mind that TFTP has it's own root directory (usual /tftpboot). You can simply copy the configuration file to this directory and the use the file name without any path. For more information about TFTP use "man tftpd".

```
$ ./bdisetup -c -p/dev/ttyS0 -b115 \
> -i151.120.25.102 \
> -h151.120.25.112 \
> -fe:/bdi3000/mytarget.cfg
Connecting to BDI loader
Writing network configuration
Configuration passed
```

5. Check configuration and exit loader mode:
The BDI is in loader mode when there is no valid firmware loaded or you connect to it with the setup tool. While in loader mode, the Mode LED is blinking. The BDI will not respond to network requests while in loader mode. To exit loader mode, the "bdisetup -v -s" can be used. You may also power-off the BDI, wait some time (1min.) and power-on it again to exit loader mode.

```
$ ./bdisetup -v -p/dev/ttyS0 -b115 -s
BDI Type : BDI3000 (SN: 30000154)
Loader   : V1.00
Firmware : V1.00 bdiGDB for ARM
MAC      : 00-0c-01-30-00-01
IP Addr  : 151.120.25.102
Subnet   : 255.255.255.255
Gateway  : 255.255.255.255
Host IP  : 151.120.25.112
Config   : /bdi3000/mytarget.cfg
```

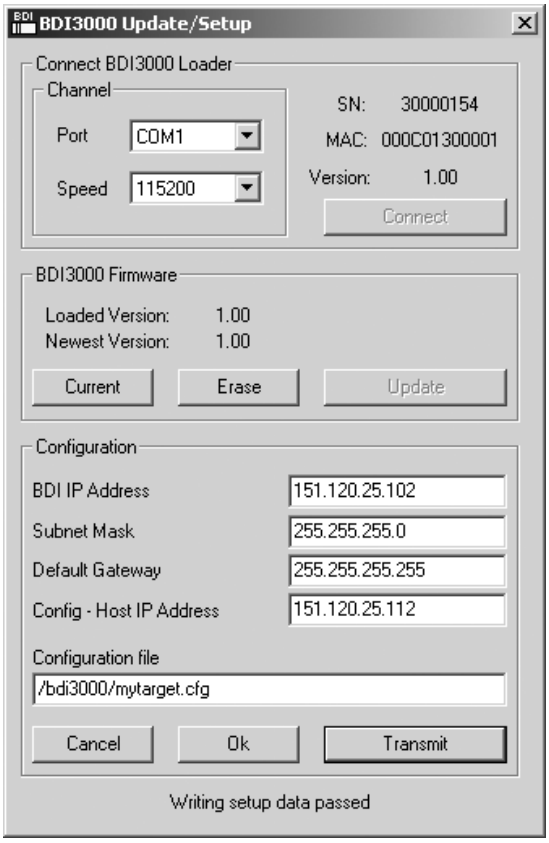
The Mode LED should go off, and you can try to connect to the BDI via Telnet.

```
$ telnet 151.120.25.102
```


2.5.2 Configuration with a Windows host

First make sure that the BDI is properly connected (see Chapter 2.1 to 2.4).

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.



dialog box «BDI3000 Update/Setup»

Before you can use the BDI3000 together with the GNU debugger, you must store the initial configuration parameters in the BDI3000 flash memory. The following options allow you to do this:

Port	Select the communication port where the BDI3000 is connected during this setup session. If you select Network, make sure the Loader is already active (Mode LED blinking). If there is already a firmware loaded and running, use the Telnet command "boot loader" to activate Loader Mode.
Speed	Select the baudrate used to communicate with the BDI3000 loader during this setup session.
Connect	Click on this button to establish a connection with the BDI3000 loader. Once connected, the BDI3000 remains in loader mode until it is restarted or this dialog box is closed.
Current	Press this button to read back the current loaded BDI3000 firmware version. The current firmware version will be displayed.

Erase	Press this button to erase the current loaded firmware.
Update	This button is only active if there is a newer firmware version present in the execution directory of the bdiGDB setup software. Press this button to write the new firmware into the BDI3000 flash memory.
BDI IP Address	Enter the IP address for the BDI3000. Use the following format: xxx.xxx.xxx.xxx e.g.151.120.25.101 Ask your network administrator for assigning an IP address to this BDI3000. Every BDI3000 in your network needs a different IP address.
Subnet Mask	Enter the subnet mask of the network where the BDI is connected to. Use the following format: xxx.xxx.xxx.xxxe.g.255.255.255.0 A subnet mask of 255.255.255.255 disables the gateway feature. Ask your network administrator for the correct subnet mask.
Default Gateway	Enter the IP address of the default gateway. Ask your network administrator for the correct gateway IP address. If the gateway feature is disabled, you may enter 255.255.255.255 or any other value.
Config - Host IP Address	Enter the IP address of the host with the configuration file. The configuration file is automatically read by the BDI3000 after every start-up.
Configuration file	Enter the full path and name of the configuration file. This name is transmitted to the TFTP server when reading the configuration file.
Transmit	Click on this button to store the configuration in the BDI3000 flash memory.

Note:
Using this setup tool via the Network channel is only possible if the BDI3000 is already in Loader mode (Mode LED blinking). To force Loader mode, enter "boot loader" at the Telnet. The setup tool tries first to establish a connection to the Loader via the IP address present in the "BDI IP Address" entry field. If there is no connection established after a time-out, it tries to connect to the default IP (192.168.53.72).

2.5.3 Configuration via Telnet / TFTP

The firmware update and the initial configuration of the BDI3000 can also be done interactively via a Telnet connection and a running TFTP server on the host with the firmware file. In cases where it is not possible to connect to the default IP, the initial setup has to be done via a serial connection.

To avoid data line conflicts, the BDI3000 must be disconnected from the target system while programming the firmware for an other target CPU family.

Following the steps to bring-up a new BDI3000 or updating the firmware.
Connect to the BDI Loader via Telnet.
If a firmware is already running enter "boot loader" and reconnect via Telnet.

```
$ telnet 192.168.53.72
or
$ telnet <your BDI IP address>
```

Update the network parameters so it matches your needs:

```
LDR>network
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 192.168.53.72
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 255.255.255.255
    Config File :

LDR>netip 151.120.25.102
LDR>nethost 151.120.25.112
LDR>netfile /bdi3000/mytarget.cfg


LDR>network
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg

LDR>network save
saving network configuration ... passed
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg
```

In case the subnet has changed, reboot before trying to load the firmware

```
LDR>boot loader
```


Connect again via Telnet and program the firmware into the BDI flash:

```
$ telnet 151.120.25.102

LDR>info
    BDI Firmware: not loaded
    BDI CPLD ID : 01285043
    BDI CPLD UES: ffffffff
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg

LDR>fwload e:/temp/b30armgd.100
erasing firmware flash ... passed
programming firmware flash ... passed

LDR>info
    BDI Firmware: 14 / 1.00
    BDI CPLD ID : 01285043
    BDI CPLD UES: ffffffff
    BDI MAC     : 00-0c-01-30-00-01
    BDI IP      : 151.120.25.102
    BDI Subnet  : 255.255.255.0
    BDI Gateway : 255.255.255.255
    Config IP   : 151.120.25.112
    Config File : /bdi3000/mytarget.cfg
LDR>
```

To boot now into the firmware use:

```
LDR>boot
```

The Mode LED should go off, and you can try to connect to the BDI again via Telnet.

```
telnet 151.120.25.102
```


2.6 Testing the BDI3000 to host connection

After the initial setup is done, you can test the communication between the host and the BDI3000. There is no need for a target configuration file and no TFTP server is needed on the host.

- If not already done, connect the BDI3000 system to the network.
- Power-up the BDI3000.
- Start a Telnet client on the host and connect to the BDI3000 (the IP address you entered during initial configuration).
- If everything is okay, a sign on message like «BDI Debugger for Embedded PowerPC» and a list of the available commands should be displayed in the Telnet window.

2.7 TFTP server for Windows

The bdiGDB system uses TFTP to access the configuration file and to load the application program. Because there is no TFTP server bundled with Windows, Abatron provides a TFTP server application **tftpsrv.exe**. This WIN32 console application runs as normal user application (not as a system service).

Command line syntax: tftpsrv [p] [w] [dRootDirectory]

Without any parameter, the server starts in read-only mode. This means, only read access request from the client are granted. This is the normal working mode. The bdiGDB system needs only read access to the configuration and program files.

The parameter [p] enables protocol output to the console window. Try it.
The parameter [w] enables write accesses to the host file system.
The parameter [d] allows to define a root directory.

`tftpsrv p`	Starts the TFTP server and enables protocol output
`tftpsrv p w`	Starts the TFTP server, enables protocol output and write accesses are allowed.
`tftpsrv dC:\tftp\`	Starts the TFTP server and allows only access to files in C:\tftp and its subdirectories. As file name, use relative names. For example "bdi\mpc750.cfg" accesses "C:\tftp\bdi\mpc750.cfg"

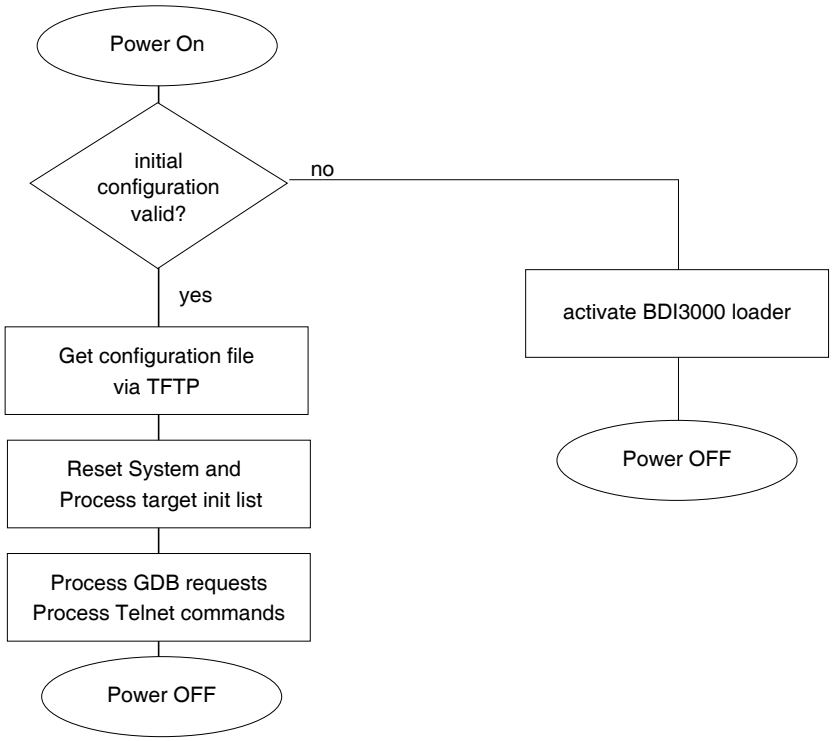
You may enter the TFTP server into the Startup group so the server is started every time you login.

3 Using bdiGDB

3.1 Principle of operation

The firmware within the BDI handles the GDB request and accesses the target memory or registers via the JTAG interface. There is no need for any debug software on the target system. After loading the code via TFTP debugging can begin at the very first assembler statement.

Whenever the BDI system is powered-up the following sequence starts:



Breakpoints:
There are two breakpoint modes supported. One of them (SOFT) is implemented by replacing application code with a special pattern. The other (HARD) uses the built in breakpoint logic. If HARD is used, only up to 2 breakpoints can be active at the same time.
The following example selects SOFT as the breakpoint mode:

```
BREAKMODE   SOFT      ;SOFT or HARD, HARD uses hardware breakpoints
```

All the time the application is suspended (i.e. caused by a breakpoint) the target processor remains in debug mode.

3.2 Configuration File

The configuration file is automatically read by the BDI3000 after every power on.
The syntax of this file is as follows:

```
; comment
[part name]
core# identifier parameter1 parameter2 ..... parameterN  ; comment
core# identifier parameter1 parameter2 ..... parameterN
.....
[part name]
core# identifier parameter1 parameter2 ..... parameterN
core# identifier parameter1 parameter2 ..... parameterN
.....
             etc.
```

Numeric parameters can be entered as decimal (e.g. 700) or as hexadecimal (0x80000).

The core# is optional. If not present the BDI assume core #0. See also chapter "Multi-Core Support".

3.2.1 Part [INIT]

The part [INIT] defines a list of commands which should be executed every time the target comes out of reset. The commands are used to get the target ready for loading the program file.

WGPR register value — Write value to the selected general purpose register.
- register — the register number 0 .. 15
- value — the value to write into the register

Example: WGPR 0 5

WREG name value — Write value to the selected register/memory by name
- name — the case sensitive register name from the reg def file
- value — the value to write to the register/memory

Example: WREG cpsr 0x600000D3

WCP15 register value — Write value to the selected Coprocessor 15 register.
- register — the register number (see chapter CP15 registers)
- value — the value to write into the register

Example: WCP15 2 0x00004000 ; set Translation Base Address

WM8 address value — Write a byte (8bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM8 0xFFFFFA21 0x04 ; SYPCR: watchdog disable ...

WM16 address value — Write a half word (16bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM16 0x02200200 0x0002 ; TBSCR

WM32 address value — Write a word (32bit) to the selected memory place.
- address — the memory address
- value — the value to write to the target memory

Example: WM32 0x02200000 0x01632440 ; SIUMCR

WBIN address filename — Write a binary image to the selected memory place. The binary image is read via TFTP from the host. Up to 4 such entries are supported.
- address — the memory address
- filename — the filename including the full path
- Example: WBIN 0x4000 pagetable.bin

DELAY value — Delay for the selected time.
- value — the delay time in milliseconds (1...30000)

Example: DELAY 500 ; delay for 0.5 seconds

RM8 address [xor]

Read a byte (8bit) from the selected memory place.

RM16 address [xor]

Read a half word (16bit) from the selected memory place.

RM32 address [xor]

Read a word (32bit) from the selected memory place.

address	the memory address
xor	optional XOR pattern applied to the read value
Example:	RM32 0x00000000

WMX and or

Writes back a modified read value. The address and size is the same as used by RM8, RM16 or RM32. This allows simple bit manipulations.

and	the AND pattern applied to the read value
or	the OR pattern applied to the read value
Example:	RM32 0x200000000 0x10101010 ; read and XOR WMX 0xff00ff00 0x00000003 ; AND, OR and write back

WAIT mask equal

Waits until ((memory & mask) == equal). The last RM8, RM16 or RM32 entry defines the address and the size for the following WAIT.

mask	the bit mask used before comparing
equal	the value to compare against
Example:	RM16 0x2000000a WAIT 0x000f0ff 0x00001034 ; wait until equal

MMAP start end

Because a memory access to an invalid memory space via JTAG leads to a deadlock, this entry can be used to define up to 32 valid memory ranges. If at least one memory range is defined, the BDI checks against this range(s) and avoids accessing of not mapped memory ranges.

start	the start address of a valid memory range
end	the end address of this memory range

Example: MMAP 0xFFE00000 0xFFFFFFFF ;Boot ROM

EXEC addr [time]

This entry cause the processor to start executing the code at addr. The optional second parameter defines a time in us how long the BDI let the processor run until it is halted. By default the BDI let it run for 500 us.This EXEC function maybe used to access CP15 registers that are not directly accessible via JTAG.

addr	the start address of the code to execute
time	the time the BDI let the processor run (micro seconds).
Example:	EXEC 0x40000000 ; execute code

CLOCK value

This entry allows to change the JTAG clock frequency during processing of the init list. But the final JTAG clock after processing the init list is taken from the CLOCK entry in the [TARGET] section. This entry maybe of interest to speed-up JTAG clock as soon as possible (after PLL setup).

value	see CLOCK parameter in [TARGET] section
Example:	CLOCK 2 ; switch to 16 MHz JTAG clock

Using a startup program to initialize the target system:
For targets where initialization can not be done with a simple initialization list, there is the possibility to download and execute a special startup code. The startup code must be present in a file on the host. The last instruction in this startup code should be a SWI. After processing the initlist, the BDI downloads this startup code to RAM, starts it and waits until it completes. If there is no SWI instruction in the startup code, the BDI terminates it after a timeout of 5 seconds.

FILE filename — The name of the file with the startup code. This name is used to access the startup code via TFTP.
- filename — the filename including the full path
- Example: FILE F:\gdb\target\config\pid7t\startup.hex

FORMAT format — The format of the startup file. Currently COFF, S-Record, a.out, Binary and ELF file formats are supported. If the startup code is already stored in ROM on the target, select ROM as the format.
- format — COFF, SREC, AOUT, BIN, ELF or ROM
- Example: FORMAT COFF

START address — The address where to start the startup code. If this value is not defined and the core is not in ROM, the address is taken from the code file. If this value is not defined and the core is already in ROM, the PC will not be set before starting the code.
- address — the address where to start the startup code
- Example: START 0x10000

Note:
If an init list and a startup code file are present, the init list is processed first and then the startup code is loaded and executed. Therefore it is possible first to enable some RAM with the init list before the startup code is loaded and executed.

```
[INIT]
WM32        0x0B000020  0x00000000  ;Clear Reset Map

FILE        d:\gdb\bdi\startup.hex
FORMAT      SREC
START       0x100
```


3.2.2 Part [TARGET]

The part [TARGET] defines some target specific values.

CPUTYPE type

This value gives the BDI information about the connected CPU.

type The CPU type from the following list:
ARM7TDMI, ARM7DI, ARM710T,ARM720T,ARM740T
ARM9TDMI, ARM920T, ARM940T, TMS470
ARM9E, ARM946E, ARM966E, ARM926E
TI925T, MAC7100, FA526
FERO926, FERO946, FERO966

Example: CPUTYPE ARM920T

CLOCK main [init]

With this value(s) you can select the JTAG clock rate the BDI3000 uses when communication with the target CPU. The "main" entry is used after processing the initialization list. The "init" value is used after target reset until the initialization list is processed. If there is no "init" value defined, the "main" value is used all the times.
Adaptive clocking needs a special target connector cable.

main,init: The clock frequency in Hertz or an index value from the following table:

0 = Adaptive

1 = 32 MHz	7 = 1 MHz	13 = 10 kHz
2 = 16 MHz	8 = 500 kHz	14 = 5 kHz
3 = 11 MHz	9 = 200 kHz	15 = 2 kHz
4 = 8 MHz	10 = 100 kHz	16 = 1 kHz
5 = 5 MHz	11 = 50 kHz	
6 = 4 MHz	12 = 20 kHz	

```
Example: CLOCK 2          ; 16 MHz JTAG clock
         CLOCK 8000000    ;  8 MHz JTAG clock
```

POWERUP delay

The value entered in this configuration line is the delay time in milliseconds the BDI waits before it begins the reset sequence. This time should be longer than the on-board reset circuit asserts RESET (default is 1 seconds). If RESET type is not NONE, the BDI asserts the reset signal via the debug connector as soon as power-up is detected.

delay the power-up start delay in milliseconds

```
Example: POWERUP  5000    ;start delay after power-up
```

RESET type [time]

Normally the BDI drives the reset line during startup. If reset type is NONE, the BDI does not assert a hardware reset during startup. This entry can also be used to change the default reset time.

type NONE
ONCE (don't monitor reset line)
HARD (default)
SGOLD (enables S-GOLD ARM9 TAP during reset)

time The time in milliseconds the BDI assert the reset signal.

```
Example: RESET NONE ; no reset during startup
         RESET HARD 1000 ; assert RESET for 1 second
```


TRST type

Normally the BDI uses an open drain driver for the TRST signal. This is in accordance with the ARM recommendation. For boards where TRST is simply pulled low with a weak resistor, TRST will always be asserted and JTAG debugging is impossible. In that case, the TRST driver type can be changed to push-pull. Then the BDI actively drives also high level.

type	OPENDRAIN (default) PUSHPULL
Example:	TRST PUSHPULL ; Drive TRST also high

STARTUP mode [runtime]

This parameter selects the target startup mode. The following modes are supported:

RESET	This default mode forces the target to debug mode immediately out of reset. No code is executed after reset.
STOP	In this mode, the BDI lets the target execute code for "runtime" milliseconds after reset. This mode is useful when monitor code should initialize the target system.
RUN	After reset, the target executes code until stopped by the Telnet "halt" command.
Example:	STARTUP STOP 3000 ; let the CPU run for 3 seconds

WAKEUP time

This entry in the init list allows to define a delay time (in ms) the BDI inserts between releasing the reset line and starting communicating with the target. This delay is necessary when a target needs some wake-up time after a reset (e.g. Cirrus EP7209).

time	the delay time in milliseconds
Example:	WAKEUP 3000 ; insert 3sec wake-up time

BDIMODE mode param

This parameter selects the BDI debugging mode. The following modes are supported:

LOADONLY	Loads and starts the application code. No debugging via JTAG interface.
AGENT	The debug agent runs within the BDI. There is no need for any debug software on the target. This mode accepts a second parameter. If RUN is entered as a second parameter, the loaded application will be started immediately, otherwise only the PC is set and BDI waits for GDB requests.
Example:	BDIMODE AGENT RUN

ENDIAN format

This entry defines the endiannes of the memory system.

format	The endiannes of the target memory: LITTLE (default) BIG
Example:	ENDIAN LITTLE

VECTOR CATCH [mask] When this line is present, the BDI catches exceptions. For ARM7 targets or when there is no mask value present, catching exceptions is only possible if the memory at address 0x00000000 to 0x0000001F is writable. For ARM9 targets, the mask is used to setup the EmbeddedICE Vector catch register. Do not define a mask for ARM7 targets.

mask	ARM9 only, selects the exceptions to catch
Example:	VECTOR CATCH ; catch all unhandled exception VECTOR CATCH 0x1F ;catch Abort, SWI, Undef, Reset

BREAKMODE mode [opc] This parameter defines how breakpoints are implemented and optional the breakpoint opcode to use (See also 3.3.3 Breakpoint Handling).

mode = SOFT	This is the normal mode. Breakpoints are implemented by replacing code with a special opcode.
mode = HARD	In this mode, the breakpoint hardware is used. Only 2 breakpoints at a time are supported.
opc	Defines the opcode the BDI should use for a breakpoint. The recommended opcode is 0xDFFFDFFF. This opcode allows to debug mixed ARM/Thumb applications. **Note:** For ARM9E cores, the BKPT instruction is used for software breakpoints unless 0xDFFFDFFF is defined. This in order to override the use of BKPT.
Example:	BREAKMODE HARD BREAKMODE SOFT 0xdfffdfff

STEPMODE mode This parameter defines how single step (instruction step) is implemented. The alternate step mode (HWBP) may be useful when stepping instructions should not enter exception handling.

JTAG	This is the default mode. For ARM9 targets, the JTAG single step feature is used. For ARM7 targets, a range breakpoint that excludes the current instruction is used.
HWBP	In this mode, a hardware breakpoint on the next instruction(s) is used to implement single stepping.
Example:	STEPMODE HWBP

WORKSPACE address If a workspace is defined, the BDI uses a faster download mode via the ARM's Debugger Communications Channel (DCC). The workspace is used for a short code sequence that reads from the DDC and writes to memory. There must be at least 32 bytes of RAM available for this code. There is no handshake between the BDI and the code consuming the data transferred via DCC. If the helper code on the target executes to slow, this download mode may fail and you have to disable it.

address	the address of the RAM area
Example:	WORKSPACE 0x00000020

SIO port [baudrate]

When this line is present, a TCP/IP channel is routed to the BDI's RS232 connector. The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). On the host, open a Telnet session using this port. Now you should see the UART output in this Telnet session. You can use the normal Telnet connection to the BDI in parallel, they work completely independent. Also input to the UART is implemented.

port	The TCP/IP port used for the host communication.
baudrate	The BDI supports 2400 ... 115200 baud
Example:	SIO 7 9600 ;TCP port for virtual IO

DCC port

When this line is present, a TCP/IP channel is routed to the ARM debug communication channel (DCC). The port parameter defines the TCP port used for this BDI to host communication. You may choose any port except 0 and the default Telnet port (23). On the host, open a Telnet session using this port. Now you should see the DCC output in this Telnet session. You can use the normal Telnet connection to the BDI in parallel, they work completely independent. Also input to DCC is implemented.

port	The TCP/IP port used for the host communication.
Example:	DCC 7 ;TCP port for DCC I/O

Daisy chained JTAG devices:
For ARM targets, the BDI can also handle systems with multiple devices connected to the JTAG scan chain. In order to put the other devices into BYPASS mode and to count for the additional bypass registers, the BDI needs some information about the scan chain layout. Enter the number (count) and total instruction register (irlen) length of the devices present before the ARM chip (Predecessor). Enter the appropriate information also for the devices following the ARM chip (Successor):

SCANPRED count irlen

This value gives the BDI information about JTAG devices present before the ARM chip in the JTAG scan chain.

count	The number of preceding devices
irlen	The sum of the length of all preceding instruction registers (IR).
Example:	SCANPRED 1 8 ; one device with an IR length of 8

SCANSUCC count irlen

This value gives the BDI information about JTAG devices present after the ARM chip in the JTAG scan chain.

count	The number of succeeding devices
irlen	The sum of the length of all succeeding instruction registers (IR).
Example:	SCANSUCC 2 12 ; two device with an IR length of 8+4

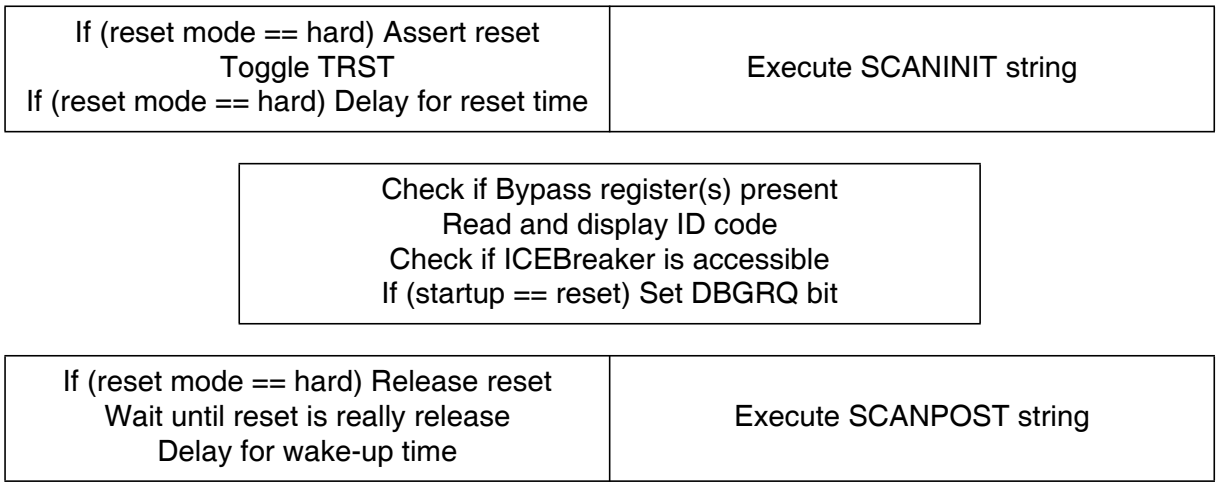
Low level JTAG scan chain configuration:
Sometimes it is necessary to configure the test access port (TAP) of the target before the ARM debug interface is visible and accessible in the usual way. The BDI supports this configuration in a very generic way via the SCANINIT and SCANPOST configuration commands. Both accept a string that defines the JTAG sequences to execute. The following example shows how to use these commands:

```
; Configure ICEPick module to make ARM926 TAP visible
SCANINIT    t1:w1000:t0:w1000:     ;toggle TRST
SCANINIT    i6=07:d8=89:i6=02:     ;connect and select router
SCANINIT    d32=81000082:          ;set IP control
SCANINIT    d32=a018206f:          ;configure TAP0
SCANINIT    d32=a018216f:cl5:      ;enable TAP0, clock 5 times in RTI
SCANINIT    i10=ffff               ;scan bypass
;
; Between SCANINIT and SCANPOST the ARM ICEBreaker is configured
; and the DBGRQ bit in the ARM debug control register is set.
;
SCANPOST    i10=002f:              ;IP(router) - ARM(bypass)
SCANPOST    d33=0102000106:        ;IP control = SysReset
SCANPOST    i10=ffff               ;scan bypass
```

The following low level JTAG commands are supported in the string. Use ":" between commands.

```
I<n>=<...b2b1b0>   write IR, b0 is first scanned
D<n>=<...b2b1b0>   write DR, b0 is first scanned
                     n  : the number of bits 1..256
                     bx : a data byte, two hex digits
W<n>               wait for n (decimal) micro seconds
T1                 assert TRST
T0                 release TRST
R1                 assert RESET
R0                 release RESET
CH<n>              clock TCK n (decimal) times with TMS high
CL<n>              clock TCK n (decimal) times with TMS low
```

The following diagram shows the parts of the standard reset sequence that are replaced with the SCAN string. Only the appropriate part of the reset sequence is replaced. If only a SCANINIT string is defined, then the standard "post" sequence is still executed.



3.2.3 Part [HOST]

The part [HOST] defines some host specific values.

IP ipaddress

The IP address of the host.

ipaddress	the IP address in the form xxx.xxx.xxx.xxx
Example:	IP 151.120.25.100

FILE filename

The default name of the file that is loaded into RAM using the Telnet 'load' command. This name is used to access the file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name.

filename	the filename including the full path or $ for relative path.
Example:	FILE F:\gnu\demo\arm\test.elf
	FILE $test.elf

FORMAT format [offset]

The format of the image file and an optional load address offset. If the image is already stored in ROM on the target, select ROM as the format. The optional parameter "offset" is added to any load address read from the image file.

format	SREC, BIN, AOUT, ELF, COFF or ROM
Example:	FORMAT ELF
	FORMAT ELF 0x10000

LOAD mode

In Agent mode, this parameters defines if the code is loaded automatically after every reset.

mode	AUTO, MANUAL
Example:	LOAD MANUAL

START address

The address where to start the program file. If this value is not defined and the core is not in ROM, the address is taken from the code file. If this value is not defined and the core is already in ROM, the PC will not be set before starting the target. This means, the program starts at the normal reset address (0x00000000).

address	the address where to start the program file
Example:	START 0x10000

DEBUGPORT port [RECONNECT]

The TCP port GDB uses to access the target. If the RECONNECT parameter is present, an open TCP/IP connection (Telnet/GDB) will be closed if there is a connect request from the same host (same IP address).

port	the TCP port number (default = 2001)
Example:	DEBUGPORT 2001

PROMPT string

This entry defines a new Telnet prompt. The current prompt can also be changed via the Telnet interface.

Example:	PROMPT AT91>

DUMP filename

The default file name used for the Telnet DUMP command.

filename	the filename including the full path
Example:	DUMP dump.bin

TELNET mode

By default the BDI sends echoes for the received characters and supports command history and line editing. If it should not send echoes and let the Telnet client in "line mode", add this entry to the configuration file.

mode	ECHO (default), NOECHO or LINE
Example:	TELNET NOECHO ; use old line mode

3.2.4 Part [FLASH]

The Telnet interface supports programming and erasing of flash memories. The bdiGDB system has to know which type of flash is used, how the chip(s) are connected to the CPU and which sectors to erase in case the ERASE command is entered without any parameter.

CHIPTYPE type [fsys]

This parameter defines the type of flash used. It is used to select the correct programming algorithm.

format	AM29F, AM29BX8, AM29BX16, I28BX8, I28BX16, AT49, AT49X8, AT49X16, STRATAX8, STRATAX16, MIRROR, MIRRORX8, MIRRORX16, S29M64X8, S29M32X16, S29WSRX16 M58X32, AM29DX16, AM29DX32, CFM32, CFM16, LPC2000, STA2051, STR710F, ST30F, ADUC7000, AT91SAM7S, STR910F, SAM9XE
fsys	For the CMF32, CMF16 and LPC2000, the BDI needs to know the system frequency. Enter the correct value for fsys in kHz.
Example:	CHIPTYPE AM29F CHIPTYPE CFM32 8000 ; fsys is 8 MHz

CHIPSIZE size

The size of **one** flash chip in bytes (e.g. AM29F010 = 0x20000). This value is used to calculate the starting address of the current flash memory bank.

size	the size of one flash chip in bytes
Example:	CHIPSIZE 0x80000

BUSWIDTH width

Enter the width of the memory bus that leads to the flash chips. Do not enter the width of the flash chip itself. The parameter CHIPTYPE carries the information about the number of data lines connected to one flash chip. For example, enter 16 if you are using two AM29F010 to build a 16bit flash memory bank.

with	the width of the flash memory bus in bits (8 | 16 | 32)
Example:	BUSWIDTH 16

FILE filename

The default name of the file that is programmed into flash using the Telnet 'prog' command. This name is used to access the file via TFTP. If the filename starts with a $, this $ is replace with the path of the configuration file name. This name may be overridden interactively at the Telnet interface.

filename	the filename including the full path or $ for relative path.
Example:	FILE F:\gnu\arm\bootrom.hex FILE $bootrom.hex

FORMAT format [offset]

The format of the file and an optional address offset. The optional parameter "offset" is added to any load address read from the program file.

format	SREC, BIN, AOUT, ELF or COFF
Example:	FORMAT SREC FORMAT ELF 0x10000

WORKSPACE address

If a workspace is defined, the BDI uses a faster programming algorithm that runs out of RAM on the target system. Otherwise, the algorithm is processed within the BDI. The workspace is used for a 1kByte data buffer and to store the algorithm code. There must be at least 2kBytes of RAM available for this purpose.

address	the address of the RAM area
Example:	WORKSPACE 0x00000000

ERASE addr [increment count] [mode [wait]]

The flash memory may be individually erased or unlocked via the Telnet interface. In order to make erasing of multiple flash sectors easier, you can enter an erase list. All entries in the erase list will be processed if you enter ERASE at the Telnet prompt without any parameter. This list is also used if you enter UNLOCK at the Telnet without any parameters. With the "increment" and "count" option you can erase multiple equal sized sectors with one entry in the erase list.

address	Address of the flash sector, block or chip to erase
increment	If present, the address offset to the next flash sector
count	If present, the number of equal sized sectors to erase
mode	BLOCK, CHIP, UNLOCK Without this optional parameter, the BDI executes a sector erase. If supported by the chip, you can also specify a block or chip erase. If UNLOCK is defined, this entry is also part of the unlock list. This unlock list is processed if the Telnet UNLOCK command is entered without any parameters. **Note:** Chip erase does not work for large chips because the BDI time-outs after 3 minutes. Use block erase.
wait	The wait time in ms is only used for the unlock mode. After starting the flash unlock, the BDI waits until it processes the next entry.
Example:	ERASE 0xff040000 ;erase sector 4 of flash ERASE 0xff060000 ;erase sector 6 of flash ERASE 0xff000000 CHIP ;erase whole chip(s) ERASE 0xff010000 UNLOCK 100 ;unlock, wait 100ms ERASE 0xff000000 0x10000 7 ; erase 7 sectors

RECOVER clkd

If this entry is present, the BDI automatically executes a "JTAG lockout recovery" during reset processing if the MAC7100 flash is secured. Use this entry only if you really need to recover a secured a MAC7100 device.

clkd	The value for the CFMCLKD register used during the JTAG lockout recovery. Calculate this entry based on the reset system frequency (PLL disabled).
Example:	RECOVER 19 ; CLKD for 8 MHz system clock

MAC7110 Internal Flash:

```
WORKSPACE   0x40000000        ;workspace in internal SRAM
CHIPTYPE    CFM32 8000        ;select Program flash, fsys = 8MHz
CHIPSIZE    0x80000           ;512k internal program flash
BUSWIDTH    32                ;32-bit bus
FILE        mac7100.cfg       ;The file to program
FORMAT      BIN 0xfc101000
ERASE       0xfc100000 PAGE   ;erase page 0 (security byte will be restored)
ERASE       0xfc101000 PAGE   ;erase page 1
ERASE       0xfc102000 PAGE   ;erase page 2
ERASE       0xfc103000 PAGE   ;erase page 3
;ERASE       0xfc100000 MASS   ;mass erase (security byte will be restored)
```

LPC2000 Internal Flash:

The LPC2100 (LPC2000) internal flash is programmed using the LPC2100 built-in flash programming driver via the so called IAP Commands. Details about the IAP commands you find in the LPC2100 user's manual. This driver needs the current System Clock Frequency (CCLK) in KHz. This frequency has to be provided via the CHIPTYPE parameter:

```
CHIPTYPE LPC2000 <fsys(kHz)>
CHIPTYPE LPC2000 14745 ;select LPC2100 flash, fsys = 14.745MHz
```

The erase parameter has a different meaning. It is not an address but a bit map of the sectors to erase (bit0 = erase sector 0, bit1 = erase ....). If you add BLANK after the sector map, then a blank check is executed after the erase. Following some examples:

```
ERASE 0x000000F0 BLANK         ;erase sector 4...7 with blank check
ERASE 0x00007FFF BLANK         ;erase sector 0...14 with blank check
ERASE 0x00000002               ;erase only sector 1, no blank check
```

The BDI needs also a workspace of 2k bytes in the internal SRAM. It is used to store the data to program and to create a context from which the flash drivers can be called.

```
[FLASH]
CHIPTYPE    LPC2000 14745       ;select LPC2100 flash, fsys = 14.745MHz
CHIPSIZE    0x20000             ;128k internal flash
WORKSPACE   0x40001000          ;internal SRAM for buffer, code and stack
FILE        E:\cygwin\home\bdidemo\arm\lpc2100.bin
FORMAT      BIN 0x00000000
ERASE       0x00007FFF BLANK    ;erase sector 0...14 with blank check
```

For LPC213x/LPC214x define always 0x80000 as CHIPSIZE, independent of the actual implemented flash memory size. Based on this CHIPSSIZE the BDI selects the correct sector table.

```
[FLASH]
CHIPTYPE    LPC2000 12000       ;select LPC2100 flash, fsys = 12.000 MHz
CHIPSIZE    0x80000             ;select LPC213x/4x sector layout
WORKSPACE   0x40001000          ;internal SRAM for buffer, code and stack
FILE        E:\cygwin\home\bdidemo\arm\mcb2130.bin
FORMAT      BIN 0x00000000
ERASE       0x007FFFFF BLANK    ;erase sector 0...26 with blank check
```


STA2051/ STR710F Internal Flash:

For the STA2051 / STR710F internal flash, the BDI assumes the following structure of the address.

14 bit	2 bit	8 bit	8 bit
reserved	bank 1	reserved	bank 0

Select the sectors to erase by setting the appropriate bit in the bank0 or bank1 field. You can only set bits in one bank at the same time. It is not possible to erase both banks in one step.
Following some examples:

```
ERASE       0x00000080          ;erase sector B0F7
ERASE       0x000000FF          ;erase all sectors of bank 0
ERASE       0x00030000          ;erase all sectors of bank 1


[FLASH]
WORKSPACE   0x20000000          ;workspace in internal SRAM
CHIPTYPE    STA2051             ;STA2051 internal flash
CHIPSIZE    0x40000             ;256k internal program flash
BUSWIDTH    32                  ;select 32 for this flash
FILE        $sta2051b0.bin      ;The file to program
FORMAT      BIN 0x40000000
ERASE       0x000000FF          ;erase all sectors of bank 0
```

ST30F7xx Internal Flash:

The ST30F7xx flash is handled like the STA2051 flash. The only difference is, that there exists only flash bank 0 but with 12 sectors.

```
ERASE       0x00000FFF          ;erase all sectors of bank 0
```

ADuC7000 Internal Flash:

The BDI3000 supports programming of the ADuC7000 internal flash. As second parameter for the ERASE command, PAGE (default) or MASS can be entered. Following a configuration example:

```
[FLASH]
WORKSPACE   0x00010020          ;workspace in internal SRAM
CHIPTYPE    ADUC7000            ;ADuC7000 internal flash
CHIPSIZE    0x10000             ;64k internal program flash
BUSWIDTH    16                  ;select 16 for this flash
FILE        E:\temp\aduc8k.bin
FORMAT      BIN 0x00080000
ERASE       0x88000             ;erase page
ERASE       0x88200             ;erase page
ERASE       0x88400             ;erase page
ERASE       0x88600             ;erase page
```


AT91SAM7S Internal Flash:

The BDI3000 supports programming of the Atmel AT91SAM7S internal flash. Before using any flash function it is important that the MC_FMR is programmed with the correct values for FMCN and FWS. This can be done via the initialization list. Following a configuration example:

```
[INIT]
WM32    0xFFFFFD44  0x00008000  ;Disable watchdog
WM32    0xFFFFFD08  0xA5000001  ;Enable user reset
;
; Setup PLL
WM32    0xFFFFFC20  0x00000601  ;CKGR_MOR : Enabling the Main Oscillator
DELAY   20
WM32    0xFFFFFC2C  0x10480a0e  ;CKGR_PLLR: 96.1MHz (DIV=14,MUL=72+1)
DELAY   20
WM32    0xFFFFFC30  0x00000007  ;PMC_MCKR : MCK = PLL / 2 = 48MHz
DELAY   20
;
; Setup Internal Flash for 48MHz Master Clock
WM32    0xFFFFFF60  0x00300100  ;MC_FMR: Flash mode (FWS=1,FMCN=48)
;

[TARGET]
CPUTYPE     ARM7TDMI
CLOCK       1 10        ;JTAG clock, start with a slow clock
RESET       HARD 300    ;Assert reset line for 300 ms
BREAKMODE   HARD        ;SOFT or HARD
STEPMODE    HWBP


[FLASH]
CHIPTYPE    AT91SAM7S   ;Don't forget to set MC_FMR[FMCN] and MC_FMR[FWS]
CHIPSIZE    0x10000     ;The AT91SAM7S64 has 64kB internal flash
BUSWIDTH    32          ;Use 32-bit for AT91SAM7S
FILE        at91\sam7s.bin
FORMAT      BIN 0x00100000
```

An explicit erase is not necessary because a page is automatically erased during programming. But the BDI3000 supports also erasing a page or the complete flash memory. The ERASE command supports a second parameter, PAGE (default) or CHIP can be used. A page is erased by programming it with all 0xFF. Following an example how to erase the complete flash via Telnet:

```
BDI> erase 0x00100000 chip
```

AT91SAM9XE Internal Flash:

The BDI3000 supports programming of the Atmel AT91SAM9XE internal flash. Before using any flash function it is important that the EEFC_FMR is programmed with the correct value for FWS. This can be done via the initialization list. Have a look at the at91sam9xe.cfg configuration example.

Supported standard Flash Memories:
There are currently 3 standard flash algorithm supported. The AMD, Intel and Atmel AT49 algorithm. Almost all currently available flash memories can be programmed with one of this algorithm. The flash type selects the appropriate algorithm and gives additional information about the used flash.

For 8bit only flash:	AM29F (MIRROR), I28BX8, AT49
For 8/16 bit flash in 8bit mode:	AM29BX8 (MIRRORX8), I28BX8 (STRATAX8), AT49X8
For 8/16 bit flash in 16bit mode:	AM29BX16 (MIRRORX16), I28BX16 (STRATAX16), AT49X16
For 16bit only flash:	AM29BX16, I28BX16, AT49X16, S29WSRX16
For 16/32 bit flash in 16bit mode:	AM29DX16
For 16/32 bit flash in 32bit mode:	AM29DX32
For 32bit only flash:	M58X32

Some newer Spansion MirrorBit flashes cannot be programmed with the MIRRORX16 algorithm because of the used unlock address offset. Use S29M32X16 for these flashes.

The AMD and AT49 algorithm are almost the same. The only difference is, that the AT49 algorithm does not check for the AMD status bit 5 (Exceeded Timing Limits).
Only the AMD and AT49 algorithm support chip erase. Block erase is only supported with the AT49 algorithm. If the algorithm does not support the selected mode, sector erase is performed. If the chip does not support the selected mode, erasing will fail. The erase command sequence is different only in the 6th write cycle. Depending on the selected mode, the following data is written in this cycle (see also flash data sheets): 0x10 for chip erase, 0x30 for sector erase, 0x50 for block erase.
To speed up programming of Intel Strata Flash and AMD MirrorBit Flash, an additional algorithm is implemented that makes use of the write buffer. The Intel Strata algorithm needs a workspace, otherwise the standard Intel algorithm is used.

The following table shows some examples:

Flash	x 8	x 16	x 32	Chipsize
Am29F010	AM29F	-	-	0x020000
Am29F800B	AM29BX8	AM29BX16	-	0x100000
Am29DL323C	AM29BX8	AM29BX16	-	0x400000
Am29PDL128G	-	AM29DX16	AM29DX32	0x01000000
Intel 28F032B3	I28BX8	-	-	0x400000
Intel 28F640J3A	STRATAX8	STRATAX16	-	0x800000
Intel 28F320C3	-	I28BX16	-	0x400000
AT49BV040	AT49	-	-	0x080000
AT49BV1614	AT49X8	AT49X16	-	0x200000
M58BW016BT	-	-	M58X32	0x200000
SST39VF160	-	AT49X16	-	0x200000
Am29LV320M	MIRRORX8	MIRRORX16	-	0x400000

Note:
Some Intel flash chips (e.g. 28F800C3, 28F160C3, 28F320C3) power-up with all blocks in locked state. In order to erase/program those flash chips, use the init list to unlock the appropriate blocks:

```
WM16   0xFFF00000   0x0060       unlock block 0
WM16   0xFFF00000   0x00D0
WM16   0xFFF10000   0x0060       unlock block 1
WM16   0xFFF10000   0x00D0
       ....
WM16   0xFFF00000   0xFFFF       select read mode
```

or use the Telnet "unlock" command:

```
UNLOCK [<addr> [<delay>]]
```

addr	This is the address of the sector (block) to unlock
delay	A delay time in milliseconds the BDI waits after sending the unlock command to the flash. For example, clearing all lock-bits of an Intel J3 Strata flash takes up to 0.7 seconds.

If "unlock" is used without any parameter, all sectors in the erase list with the UNLOCK option are processed.

To clear all lock-bits of an Intel J3 Strata flash use for example:

```
BDI> unlock 0xFF000000 1000
```

To erase or unlock multiple, continuous flash sectors (blocks) of the same size, the following Telnet commands can be used:

```
ERASE  <addr> <step> <count>
UNLOCK <addr> <step> <count>
```

addr	This is the address of the first sector to erase or unlock.
step	This value is added to the last used address in order to get to the next sector. In other words, this is the size of one sector in bytes.
count	The number of sectors to erase or unlock.

The following example unlocks all 256 sectors of an Intel Strata flash (28F256K3) that is mapped to 0x00000000. In case there are two flash chips to get a 32bit system, double the "step" parameter.

```
BDI> unlock 0x00000000 0x20000 256
```


3.2.5 Part [REGS]

In order to make it easier to access target registers via the Telnet interface, the BDI can read in a register definition file. In this file, the user defines a name for the register and how the BDI should access it (e.g. as memory mapped, memory mapped with offset, ...). The name of the register definition file and information for different registers type has to be defined in the configuration file.
The register name, type, address/offset/number and size are defined in a separate register definition file. This way, you can create one register definition file for a specific target processor that can be used for all possible positions of the internal memory map. You only have to change one entry in the configuration file.

An entry in the register definition file has the following syntax:

```
name   type   addr   size
```

name	The name of the register (max. 12 characters)
type	The register type

GPR	General purpose register
CP15	Coprocessor 15 register
MM	Absolute direct memory mapped register
DMM1...DMM4	Relative direct memory mapped register
IMM1...IMM4	Indirect memory mapped register

addr	The address, offset or number of the register
size	The size (8, 16, 32) of the register

The following entries are supported in the [REGS] part of the configuration file:

FILE filename

The name of the register definition file. This name is used to access the file via TFTP. The file is loaded once during BDI startup.

filename	the filename including the full path
Example:	FILE C:\bdi\regs\reg40400.def

DMMn base

This defines the base address of direct memory mapped registers. This base address is added to the individual offset of the register.

base	the base address
Example:	DMM1 0x01000

IMMn addr data

This defines the addresses of the memory mapped address and data registers of indirect memory mapped registers. The address of a IMMn register is first written to "addr" and then the register value is access using "data" as address.

addr	the address of the Address register
data	the address of the Data register
Example:	DMM1 0x04700000

Example for a register definition (AT91M40400):

Entry in the configuration file:

```
[REGS]
DMM1   0x04700000                      ;Internal Memory Map Base Address
FILE   E:\bdi\reg40400.def             ;The register definition file
```

The register definition file:

```
;name         type   addr         size
;-----------------------------------------
;
;
; External Bus Interface (EBI) Registers
;
csr0          MM     0xFFE0000032
csr1          MM     0xFFE0000432
csr2          MM     0xFFE0000832
csr3          MM     0xFFE0000c32
csr4          MM     0xFFE0001032
csr5          MM     0xFFE0001432
csr6          MM     0xFFE0001832
csr7          MM     0xFFE0001c32
rcr           MM     0xFFE0002032
mcr           MM     0xFFE0002432
```

Now the defined registers can be accessed by name via the Telnet interface:

BDI> rd csr0
BDI>rm csr0 0x01002535

Example for CP15 register definition (ARM720T):

```
;
id            CP15   0x0000       32
control       CP15   0x0001       32
ttb           CP15   0x0002       32
dac           CP15   0x0003       32
fsr           CP15   0x0005       32
far           CP15   0x0006       32
iidc          CP15   0x0007       32     ;invalidate ID cache
itlb          CP15   0x0008       32     ;invalidate TLB
itlbs         CP15   0x2008       32     ;invalidate TLB single entry
pid           CP15   0x000d       32     ;process identifier
```


3.3 Debugging with GDB

Because the target agent runs within BDI, no debug support has to be linked to your application. There is also no need for any BDI specific changes in the application sources. Your application must be fully linked because no dynamic loading is supported.

3.3.1 Target setup

Target initialization may be done at two places. First with the BDI configuration file, second within the application. The setup in the configuration file must at least enable access to the target memory where the application will be loaded. Disable the watchdog and setting the CPU clock rate should also be done with the BDI configuration file. Application specific initializations like setting the timer rate are best located in the application startup sequence.

3.3.2 Connecting to the target

As soon as the target comes out of reset, BDI initializes it and loads your application code. If RUN is selected, the application is immediately started, otherwise only the target PC is set. BDI now waits for GDB request from the debugger running on the host.

After starting the debugger, it must be connected to the remote target. This can be done with the following command at the GDB prompt:

```
(gdb)target remote bdi3000:2001
```

bdi3000	This stands for an IP address. The HOST file must have an appropriate entry. You may also use an IP address in the form xxx.xxx.xxx.xxx
2001	This is the TCP port used to communicate with the BDI

If not already suspended, this stops the execution of application code and the target CPU changes to background debug mode.

Remember, every time the application is suspended, the target CPU is freezed. During this time no hardware interrupts will be processed.

Note: For convenience, the GDB detach command triggers a target reset sequence in the BDI.

```
(gdb)...
(gdb)detach
... Wait until BDI has resetet the target and reloaded the image
(gdb)target remote bdi3000:2001
```

Note:
GDB sometimes fails to connect to the target after a reset because it tries to read an invalid stack frame. With the following init list entries you can work around this GDB startup problem:

```
WGPR    11          0x00000020  ;set frame pointer to free RAM
WM32    0x00000020  0x00000028  ;dummy stack frame
```


3.3.3 Breakpoint Handling

GDB tells the BDI to set / clear breakpoints with the Z-Packet protocol unit. The BDI will respond to this request by replacing code in memory or by setting the appropriate hardware breakpoint. The pattern used to replace memory is the one defined with the BREAKMODE parameter. It is recommended to define a pattern of 0xDFFFDFFF because this pattern allows to debug mixed ARM/Thumb applications.

The ARM IceBreaker supports two hardware breakpoints (watchpoints). For ARM7 and ARM9, one of them is used to support software breakpoints and vector catching for ARM7 targets. The other can be used for a hardware breakpoint. To make both available for hardware breakpoints, you should select BREAKMODE HARD (ARM7/9) and disable vector catching (ARM7).

For ARM9E the BKPT instruction is used to implement software breakpoints. Then no hardware breakpoint is wasted to implement software breakpoints. To disable the use of the BKPT instruction define 0xDFFFDFFF as breakpoint pattern (BREAKMODE SOFT 0xDFFFDFFF).

User controlled hardware breakpoints:
The ARM IceBreaker has a special watchpoint hardware integrated. Normally the BDI controls this hardware in response to Telnet commands (BI, BDx) or when breakpoint mode HARD is selected. Via the Telnet commands BI and BDx, you cannot access all the features of the breakpoint hardware. Therefore the BDI assumes that the user will control / setup this watchpoint hardware as soon as the appropriate Watchpoint Control register is written to. This way the debugger or the user via Telnet has full access to all features of this watchpoint hardware. When setting a watchpoint, use the following register numbers. The values will be written to the IceBreaker immediately before a target restart.

```
100 : Watchpoint 0 Address Value        110 : Watchpoint 1 Address Value
101 : Watchpoint 0 Address Mask         111 : Watchpoint 1 Address Mask
102 : Watchpoint 0 Data Value           112 : Watchpoint 1 Data Value
103 : Watchpoint 0 Data Mask            113 : Watchpoint 1 Data Mask
104 : Watchpoint 0 Control Value        114 : Watchpoint 1 Control Value
105 : Watchpoint 0 Control Mask         115 : Watchpoint 1 Control Mask

Example:

BDI> rmib 100 0x00104560
```

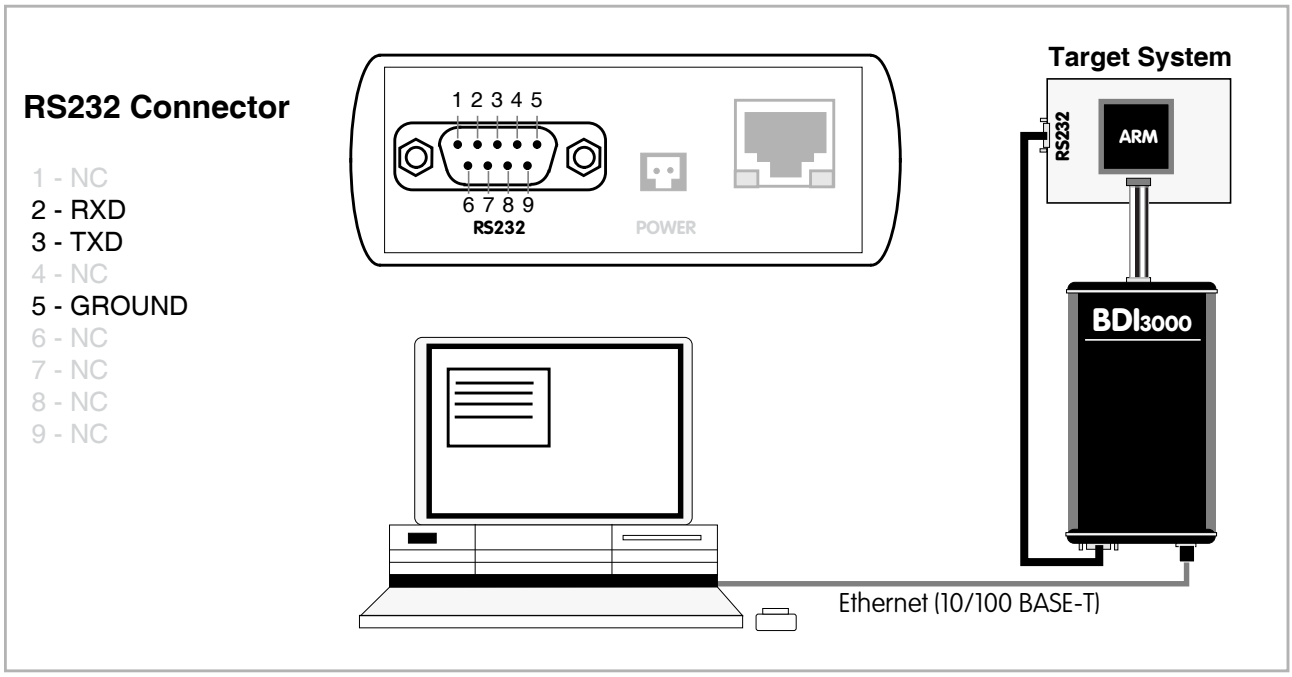
3.3.4 GDB monitor command

The BDI supports the GDB V5.x "monitor" command. Telnet commands are executed and the Telnet output is returned to GDB.

```
(gdb) target remote bdi3000:2001
Remote debugging using bdi3000:2001
0x10b2 in start ()
(gdb) monitor md 0 1
00000000 : 0xe59ff018  - 442503144  ...
```


3.3.5 Target serial I/O via BDI

A RS232 port of the target can be connected to the RS232 port of the BDI3000. This way it is possible to access the target's serial I/O via a TCP/IP channel. For example, you can connect a Telnet session to the appropriate BDI3000 port. Connecting GDB to a GDB server (stub) running on the target should also be possible.



The configuration parameter "SIO" is used to enable this serial I/O routing.
The used framing parameters are 8 data, 1 stop and not parity.

```
[TARGET]
....
SIO    7      9600          ;Enable SIO via TCP port 7 at 9600 baud
```

Warning!!!
Once SIO is enabled, connecting with the setup tool to update the firmware will fail. In this case either disable SIO first or disconnect the BDI from the LAN while updating the firmware.

3.3.6 Target DCC I/O via BDI

It is possible to route a TCP/IP port to the ARM's debug communciation channel (DCC). This way, the application running on the target can output messages via DCC that are displayed for example in a Telnet window. The BDI routes every byte received via DCC to the connected TCP/IP channel and vice versa. Below some simple functions you can link to your application in order to implement IO via DCC.

```
#define DCC_OUTPUT_BUSY 2
#define DCC_INPUT_READY 1

static unsigned int read_dcc(void) {

    unsigned int c;

    __asm__ volatile(
           "mrc p14,0, %0, c1, c0\n"
           : "=r" (c));
       return c;
}

static void write_dcc(unsigned int c) {

    __asm__ volatile(
           "mcr p14,0, %0, c1, c0\n"
           :
           : "r" (c));
}

static unsigned int poll_dcc(void) {

    unsigned int ret;

    __asm__ volatile(
           "mrc p14,0, %0, c0, c0\n"
           : "=r" (ret));
    return ret;
}

void write_dcc_char(unsigned int c) {

       while(poll_dcc() & DCC_OUTPUT_BUSY);
       write_dcc(c);
}

unsigned int read_dcc_char(void) {

       while(!(poll_dcc() & DCC_INPUT_READY));
       return read_dcc();
}


void write_dcc_string(const char* s)
{
       while (*s) write_dcc_char(*s++);
}
```


3.4 Telnet Interface

A Telnet server is integrated within the BDI. The Telnet channel is used by the BDI to output error messages and other information. Also some basic debug tasks may be done by using this interface. Enter help at the Telnet command prompt to get a list of the available commands.

Telnet Debug features:

- Display and modify memory locations
- Display and modify registers
- Single step a code sequence
- Set hardware breakpoints (for code and data accesses)
- Load a code file from any host
- Start / Stop program execution
- Programming and Erasing Flash memory

During debugging with GDB, the Telnet is mainly used to reboot the target (generate a hardware reset and reload the application code). It may be also useful during the first installation of the bdiGDB system or in case of special debug needs.

Notes:
The DUMP command uses TFTP to write a binary image to a host file. Writing via TFTP on a Linux/ Unix system is only possible if the file already exists and has public write access. Use "man tftpd" to get more information about the TFTP server on your host.

3.4.1 Command list

```
"MD    [<address>] [<count>]  display target memory as word (32bit)",
"MDH   [<address>] [<count>]  display target memory as half word (16bit)",
"MDB   [<address>] [<count>]  display target memory as byte (8bit)",
"DUMP  <addr> <size> [<file>] dump target memory to a file",
"MM    <addr> <value> [<cnt>] modify word(s) (32bit) in target memory",
"MMH   <addr> <value> [<cnt>] modify half word(s) (16bit) in target memory",
"MMB   <addr> <value> [<cnt>] modify byte(s) (8bit) in target memory",
"MT    <addr> <count>[<loop>] memory test",
"MC    [<address>] [<count>]  calculates a checksum over a memory range",
"MV                           verifies the last calculated checksum",
"RD    [<name>]               display general purpose or user defined register",
"RDUMP [<file>]               dump all user defined register to a file",
"RDALL                        display all ARM registers ",
"RDCP  <number>               display control processor 15 register",
"RDIB  [<number>]             display IceBreaker register",
"RM    {<nbr>≠<name>} <value> modify general purpose or user defined register",
"RMCP  <number>   <value>     modify control processor 15 register",
"RMIB  <number>   <value>     modify IceBreaker register",
"RESET [HALT | RUN [time]]    reset the target system, change startup mode",
"GO    [<pc>]                 set PC and start current core",
"GO    <n> <n> [<n>[<n>]]     start multiple cores in requested order",
"TI    [<pc>]                 single step an instruction",
"HALT  [<n>[<n>[<n>[<n>]]]]   force core(s) to debug mode (n = core number)",
"BI  <addr> [<mask>]          set instruction breakpoint",
"CI  [<id>]                   clear instruction breakpoint(s)",
"BD  [R|W] <addr> [<data>]    set data watchpoint (32bit access)",
"BDH [R|W] <addr> [<data>]    set data watchpoint (16bit access)",
"BDB [R|W] <addr> [<data>]    set data watchpoint ( 8bit access)",
"BDM [R|W] <addr> [<mask>]    set data watchpoint with address mask",
"CD [<id>]                    clear data watchpoint(s)",
"INFO                         display information about the current state",
"LOAD   [<offset>] [<file> [<format>]] load program file to target memory",
"VERIFY [<offset>] [<file> [<format>]] verify a program file to target memory",
"PROG   [<offset>] [<file> [<format>]] program flash memory",
"                           <format>  : SREC, BIN, AOUT, ELF or COFF",
"ERASE  [<address> [<mode>]]  erase a flash memory sector, chip or block",
"                    <mode>  : CHIP, BLOCK or SECTOR (default is sector)",
"ERASE  <addr> <step> <count> erase multiple flash sectors",
"UNLOCK [<addr> [<delay>]]    unlock a flash sector",
"UNLOCK <addr> <step> <count> unlock multiple flash sectors",
"FLASH  <type> <size> <bus>   change flash configuration",
"FENA   <addr> <size>         enable autoamtic programming to flash memory",
"FDIS                         disable autoamtic programming to flash memory",
"DELAY  <ms>                  delay for a number of milliseconds",
"SELECT <core>                change the current core",
"SCAN <nbr><len>[<...b2b1b0>] Access a JTAG scan chain, b0 is first scanned",
"                             len : the number of bits 1..256",
"                             bx  : a data byte, two hex digits",
"HOST   <ip>                  change IP address of program file host",
"PROMPT <string>              defines a new prompt string",
"CONFIG                       display or update BDI configuration",
"CONFIG <file> [<hostIP> [<bdiIP> [<gateway> [<mask>]]]]",
"UPDATE                       reload the configuration without a reboot",
"HELP                         display command list",
"BOOT  [loader]               reboot the BDI and reload the configuration",
"QUIT                         terminate the Telnet session"
```


3.4.2 CP15 Registers

Via Telnet it is possible to access the Coprocessor 15 (CP15) registers. Following the two Telnet commands that are used to access CP15 registers:

```
"RDCP  <number>               display control processor 15 register"
"RMCP  <number>   <value>     modify control processor 15 register"
```

The parameter number selects the CP15 register. This parameter has a special numbering scheme which depends of the ARM CPU type. More information is also found in the ARM documentation.

ARM710T, ARM720T,ARM740T:
The 16bit register number is used to build the appropriate MCR/MRC instruction to access the CP15 register.

```
+-----+-+-------+-------+-------+
|opc_2|0|  CRm  |0 0 0 0|  nbr  |
+-----+-+-------+-------+-------+
```

Normally opc_2 and CRm are zero and therefore you can simply enter the CP15 register number. In the register definition file "reg720t.def" you will find some examples.

ARM920T:
Via JTAG, CP15 registers are accessed either direct (physical access mode) or via interpreted MCR/MRC instructions. Read also ARM920T manual, part "Debug Support - Scan Chain 15".

Register number for physical access mode (bit 12 = 0):

```
+-----+-+-----+-+-----+-+-------+
|0 0 0|0|0 0 0|i|0 0 0|x|  nbr  |
+-----+-+-----+-+-----+-+-------+
```

The bit "i" selects the instruction cache (scan chain bit 33), the bit "x" extends access to register 15 (scan chain bit 38).

Register number for interpreted access mode (bit 12 = 1):

```
+-----+-+-------+-----+-+-------+
|opc_2|1|  CRm  |opc_1|0|  nbr  |
+-----+-+-------+-----+-+-------+
```

The 16bit register number is used to build the appropriate MCR/MRC instruction.

ARM940T, ARM946E, ARM966E:
The CP15 registers are directly accessed via JTAG.

```
+-----+-+-----+-+-----+-+-------+
|0 0 0|0|0 0 0|i|0 0 0|x|  nbr  |
+-----+-+-----+-+-----+-+-------+
```

The bit "i" selects the instruction cache (scan chain bit 32), the bit "x" extends access to register 6 (scan chain bit 37). In the register definition file "reg940t.def" you will find some examples.

ARM926E:
The 16bit register number contains the fields of the appropriate MCR/MRC instruction that would be used to access the CP15 register.

```
+-+-----+-+-----+-------+-------+
|-|opc_1|-|opc_2|  CRm  |  nbr  |
+-+-----+-+-----+-------+-------+
```

Normally opc_1, opc_2 and CRm are zero and therefore you can simply enter the CP15 register number. In the register definition file "reg926e.def" you will find some examples.

TI925T:
The CP15 registers are directly accessed via JTAG.
The following table shows the numbers used to access the CP15 registers and functions.

```
 0 (or 0x30) : ID
 1 (or 0x31) : Control
 2 (or 0x32) : Translation table base
 3 (or 0x33) : Domain access control
 5 (or 0x35) : Fault status
 6 (or 0x36) : Fault address
 8 (or 0x38) : Cache information
13 (or 0x3d) : Process ID


0x10 : TI925T Status
0x11 : TI925T Configuration
0x12 : TI925T I-max
0x13 : TI925T I-min
0x14 : TI925T Thread ID

0x18 : Flush I+D TLB
0x19 : Flush I TLB
0x1a : Flush I TLB entry
0x1b : Flush D TLB
0x1c : Flush D TLB entry

0x20 : Flush I cache
0x22 : Flush I cache entry
0x23 : Flush D cache
0x24 : Flush D cache entry address
0x25 : Clean D cache entry address
0x26 : Clean + Flush D cache entry address
0x27 : Flush D cache entry index
0x28 : Clean D cache entry index
0x29 : Clean + Flush D cache entry index
0x2a : Clean D cache
0x2b : Drain Write buffer

0x37 : I cache TLB Lock-Down
0x3a : D cache TLB Lock-Down
```


3.5 Multi-Core Support

The bdiGDB system supports concurrent debugging of up to 4 ARM cores connected to the same JTAG scan chain. For every core you can start its own GDB session. The default port numbers used to attach the remote targets are 2001 ... 2004. In the Telnet you switch between the cores with the command "select <0..3>". In the configuration file, simply begin the line with the appropriate core number. If there is no #n in front of a line, the BDI assumes core #0.

The following example defines two cores on the scan chain.

```
[TARGET]
CLOCK       1           ;JTAG clock (0=Adaptive, 1=8MHz, 2=4MHz, 3=2MHz)
WAKEUP      1000        ;wakeup time after reset

#0 CPUTYPE     ARM7TDMI
#0 SCANPRED    0 0              ;JTAG devices connected before this core
#0 SCANSUCC    1 4              ;JTAG devices connected after this core
#0 VECTOR      CATCH            ;catch unhandled exceptions
#0 BREAKMODE   SOFT 0xef180000  ;SOFT or HARD (X-Tools V1.0 break code)
#0 DCC         8                ;DCC I/O via TCP port 8

#1 CPUTYPE     ARM7TDMI
#1 SCANPRED    1 4              ;JTAG devices connected before this core
#1 SCANSUCC    0 0              ;JTAG devices connected after this core
#1 VECTOR      CATCH            ;catch unhandled exceptions
#1 BREAKMODE   SOFT 0xef180000  ;SOFT or HARD (X-Tools V1.0 break code)
#1 DCC         7                ;DCC I/O via TCP port 7
```

For a complete configuration example see "eb63_eval7t.cfg" on the diskette. This configuration was used to debug an AT91EB63 daisy chained with an Evaluator-7T board.

4 Specifications

Operating Voltage Limiting	5 VDC ± 0.25 V
Power Supply Current	typ. 500 mA max. 1000 mA
RS232 Interface: Baud Rates	9’600,19’200, 38’400, 57’600,115’200
Data Bits	8
Parity Bits	none
Stop Bits	1
Network Interface	10/100 BASE-T
BDM/JTAG clock	up to 32 MHz
Supported target voltage	1.2 – 5.0 V
Operating Temperature	+ 5 °C ... +60 °C
Storage Temperature	-20 °C ... +65 °C
Relative Humidity (noncondensing)	<90 %rF
Size	160 x 85 x 35 mm
Weight (without cables)	280 g
Host Cable length (RS232)	2.5 m
Electromagnetic Compatibility	CE compliant
Restriction of Hazardous Substances	RoHS 2002/95/EC compliant

Specifications subject to change without notice

5 Environmental notice

Disposal of the equipment must be carried out at a designated disposal site.

6 Declaration of Conformity (CE)

CE

DECLARATION OF CONFORMITY

This declaration is valid for following product:

Type of device: BDM/JTAG Interface
Product name: BDI3000

The signing authorities state, that the above mentioned equipment meets the requirements for emission and immunity according to

EMC Directive 89/336/EEC

The evaluation procedure of conformity was assured according to the following standards:

IEC 61000-6-2: 1999, mod. EN61000-6-2: 2001
IEC 61000-6-3: 1996, mod. EN61000-6-2: 2001

This declaration of conformity is based on the test report no. E1087-05-7a of Quinel, Zug, Swiss Testing Service, accreditation no. STS 037

Manufacturer:

ABATRON AG
Lettenstrasse 9
CH-6343 Rotkreuz

Authority:

Max Vock
Marketing Director

Ruedi Dummermuth
Technical Director

Rotkreuz, 7/18/2007

7 Abatron Warranty and Support Terms

7.1 Hardware

ABATRON Switzerland warrants that the Hardware shall be free from defects in material and workmanship for a period of 3 years following the date of purchase when used under normal conditions. Failure in handling which leads to defects or any self-made repair attempts are not covered under this warranty. In the event of notification within the warranty period of defects in material or workmanship, ABATRON will repair or replace the defective hardware. The customer must contact the distributor or Abatron for a RMA number prior to returning.

7.2 Software

License

Against payment of a license fee the client receives a usage license for this software product, which is not exclusive and cannot be transferred.

Copies

The client is entitled to make copies according to the number of licenses purchased. Copies exceeding this number are allowed for storage purposes as a replacement for defective storage mediums.

Update and Support

The agreement includes free software maintenance (update and support) for one year from date of purchase. After this period the client may purchase software maintenance for an additional year.

7.3 Warranty and Disclaimer

ABATRON AND ITS SUPPLIERS HEREBY DISCLAIMS AND EXCLUDES, TO THE EXTENT PERMITTED BY APPLICABLE LAW, ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.

7.4 Limitation of Liability

IN NO EVENT SHALL ABATRON OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING, WITHOUT LIMITATION, ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE HARDWARE AND/OR SOFTWARE, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, BUSINESS, DATA, GOODWILL, OR ANTICIPATED SAVINGS, EVEN IF ADVISED OF THE POSSIBILITY OF THOSE DAMAGES.

The hardware and software product with all its parts, copyrights and any other rights remain in possession of ABATRON. Any dispute, which may arise in connection with the present agreement shall be submitted to Swiss Law in the Court of Zug (Switzerland) to which both parties hereby assign competence.

Appendices

A Troubleshooting

Problem
The firmware can not be loaded.

Possible reasons

- The BDI is not correctly connected with the Host (see chapter 2).
- A wrong communication port is selected (Com 1...Com 4).
- The BDI is not powered up

Problem
No working with the target system (loading firmware is okay).

Possible reasons

- Wrong pin assignment (BDM/JTAG connector) of the target system (see chapter 2).
- Target system initialization is not correctly –> enter an appropriate target initialization list.
- An incorrect IP address was entered (BDI3000 configuration)
- BDM/JTAG signals from the target system are not correctly (short-circuit, break, ...).
- The target system is damaged.

Problem
Network processes do not function (loading the firmware was successful)

Possible reasons

- The BDI3000 is not connected or not correctly connected to the network (LAN cable or media converter)
- An incorrect IP address was entered (BDI3000 configuration)

B Maintenance

The BDI needs no special maintenance. Clean the housing with a mild detergent only. Solvents such as gasoline may damage it.

C Trademarks

All trademarks are property of their respective holders.

